User's Manual

32

RH850 Family

Type 01 Data Flash Library

Japanese Release

ZIP file name: JP_R_FDL_RH850_T01_Vx.xx_x_E

32-Bit Single-Chip Microcontroller

Supported Devices:

RH850/F1L

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corporation without notice. Please review the latest information published by Renesas Electronics Corporation through various means, including the Renesas Electronics Corporation website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.01 Sep 2013

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

NOTES FOR CMOS DEVICES

(1) VOLTAGE APPLICATION WAVEFORM AT INPUT PIN: Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (MAX) and VIH (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (MAX) and VIH (MIN).

(2) HANDLING OF UNUSED INPUT PINS: Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

(3) PRECAUTION AGAINST ESD: A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

(4) STATUS BEFORE INITIALIZATION: Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

(5) POWER ON/OFF SEQUENCE: In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current. The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

(6) INPUT OF SIGNAL DURING POWER OFF STATE : Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

How to Use This Manual

Target Readers

This manual is intended for users who wish to understand the features of the Type 01 Data Flash Library for RH850 Family and to use the library in designing and developing application systems.

Purpose

This manual is intended to give users an understanding of the correct use of the Type 01 Data Flash Library that is used for programming the data flash in RH850 Family.

Organization

This manual includes the following sections.

- Overview
- Hardware environment
- Software environment
- FDL architecture
- FDL functional specifications
- User interface
- FDL Processing Time

How to Read This Manual

It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.

- To gain a general understanding of features
  -> Read this manual in order of the table of contents.
- For details on the functions of the library
  -> Refer to section 5, User Interface, of this user's manual.

The mark <R> shows major revised points.

Conventions

Data significance: Higher-order digits to the left and lower-order digits to the right
Active low representations: $\overline{xxx}$ (overscore over pin and signal name)
Note: Footnote for item marked with **Note** in the text.
Caution: Information requiring particular attention
Remark: Supplementary information
Numeral representation: Binary ... xxxx or xxxxB
Decimal ... xxxx
Hexadecimal ... xxxxH or 0xXXXX

All trademarks and registered trademarks are the property of their respective owners.
EEPROM is a registered trademark of Renesas Electronics Corporation.

Contents

Chapter 1 Introduction 7
1.1 Overview 7
1.2 Target Devices 7
1.3 Definition of Terms 7
Chapter 2 Hardware Environment 8
2.1 Data Flash 8
2.1.1 CPU Operating Frequency Configuration 9
2.2 Notes on Hardware Environment 9
Chapter 3 Software Environment 10
3.1 FDL Sections 10
3.2 FDL Resources 10
3.3 Notes on Software Environment 10
Chapter 4 FDL Architecture 11
4.1 System Structure 11
4.1.1 FDL Pool 11
4.2 Notes on FDL Architecture 12
Chapter 5 FDL Function 13
5.1 FDL Functions / Commands of the R_FDL_Execute Function 13
5.2 List of Commands 13
5.3 Basic Flowchart 14
5.4 Flowchart for Command Operations 15
5. 4. 1 During Command Execution/During Busy 15
5.5 FDL Incorporation into the User System 16
5.6 Back Ground Operation(BGO) 17
5.7 Suspend and Resume 19
5.8 Standby and Wakeup 21
5.9 Notes on the Functional Specifications of the FDL 23
Chapter 6 User Interface 25
6.1 Runtime Settings 25
6.1.1 Note on Runtime Settings 26
6.2 Request Structure Settings 27
6.2.1 User Write Access 28
6.2.2 User Read Access 29
6.2.3 Note on Request Structure Settings 29
6.3 FDL Function Calls 30
6.4 Data Types 30
6.5 FDL Functions 31
R_FDL_Init 32
R_FDL_Execute 34
R_FDL_Handler 41
R_FDL_SuspendRequest 42
R_FDL_ResumeRequest 44

R_FDL_StandBy 46
R_FDL_GetVersionString 50
Chapter 7 FDL Processing Time 51
Appendix A Revision History 53

Chapter 1 Introduction

1.1 Overview

The Data Flash library is free software to rewrite data flash through user programs. This software offers interface for “device operations on data flash” such as:

- Erase
- Write
- Read
- Blank check

Use this manual along with the release notes included in the package of the Data Flash library and the user's manual for the target device.

1.2 Target Devices

For the latest device information, please contact our distributors or sales representatives.

1.3 Definition of Terms

The terms used in this user’s manual are defined below.

Table 1-1 Definition of Terms

Definition of Terms	Description
Type	"Type" is an identification name representing an FDL type. Use the type corresponding to your device.
FDL	An abbreviation of the data flash library used to control Data Flash Memory for storing data. The RH850 Data Flash Library Type 01 is hereafter referred to as "FDL".
FCL	An abbreviation of the code flash library used to control Code Flash Memory for storing data.
FDL function	A generic term for the functions offered by the FDL.
Block number	A number which identifies a block of Flash memory.

Chapter 2 Hardware Environment

This chapter describes the hardware environment required for the user to rewrite Data Flash by using the FDL.

2.1 Data Flash

The RH850 family's on-chip Data Flash has a "block structure". That is, it is divided into several blocks. A block is the smallest unit of erase operation on on-chip Data Flash for the FDL. Flash is always erased in units of blocks. Write and blank check operations are performed by specifying the start address and execution size. When read and write operations on data flash are performed using the FDL, the addresses required for these operations are those relative to the start (address 0) of block 0 in that memory.

Table 2-1 gives an example of Data Flash addresses when the FDL is running on a device supported by Type 01. For information about the block structure for the target device, refer to the user's manual for the target device.

Table 2-1 Data Flash Addresses at the Time of FDL Operation

Example: RH850/F1L with 32-Kbyte Data Flash Block size = 64 bytes		FDL-control address
Block No.	Address	
511	0xFF20_7FC0 ～ 0xFF20_7FFF	0x0000_7FC0 ～ 0x0000_7FFF
510	0xFF20_7F80 ～ 0xFF20_7FBF	0x0000_7F80 ～ 0x0000_7FBF
509	0xFF20_7F40 ～ 0xFF20_7F7F	0x0000_7F40 ～ 0x0000_7F7F
⋮	⋮	⋮
5	0xFF20_0140 ～ 0xFF20_017F	0x0000_0140 ～ 0x0000_017F
4	0xFF20_0100 ～ 0xFF20_013F	0x0000_0100 ～ 0x0000_013F
3	0xFF20_00C0 ～ 0xFF20_00FF	0x0000_00C0 ～ 0x0000_00FF
2	0xFF20_0080 ～ 0xFF20_00BF	0x0000_0080 ～ 0x0000_00BF
1	0xFF20_0040 ～ 0xFF20_007F	0x0000_0040 ～ 0x0000_007F
0	0xFF20_0000 ～ 0xFF20_003F	0x0000_0000 ～ 0x0000_003F

2.1.1 CPU Operating Frequency Configuration

To rewrite data flash through the FDL requires configuring the CPU frequency.
This frequency is used for timing calculation inside the FDL. For the procedure to configure the CPU operating frequency, see section 6.1, Runtime Settings. Also see table 2-2, Notes on Hardware Environment.

2.2 Notes on Hardware Environment

Below are notes on the hardware environment. For details of notes other than No.1 and No.2 below, be sure to see the user's manual for the target device.

Table 2-2 Notes on Hardware Environment

No.	Note
1	If the CPU frequency has digits below the decimal point, round it up to the nearest whole number and set this number. [Example] When there is no decimal point: Specify a CPU frequency 32 MHz as 32. [Example] When there is a decimal point: Specify a CPU frequency of 25.3 MHz as 26.
2	When read and write operations on data flash are performed with the FDL, the addresses required for these operations are those relative to the start (address 0) of block 0 in the data flash. Note that they are not absolute addresses.
3	The FDL does not perform compare check with the CPU operating frequency set by the user. The CPU frequency is a parameter depending on the user system, and so the FDL cannot check this parameter. Check the specifications of the target device and be sure to configure the correct CPU frequency. If checking of the CPU frequency is required, configure the user application to perform the checking. (The FDL checks the CPU operating frequency only within the frequency range specific to each RH850 product.)
4	Some devices have restrictions on CPU frequencies for Data Flash operation.
5	For hardware information of the Data Flash in the target device (such as a block size and unit of reading), refer to the specifications and notes described in the user's manual for the target device.
6	The FDL is only usable in supervisor mode (CPU operating mode) and cannot be used in user mode.

Chapter 3 Software Environment

This chapter describes the software environment required for the user to rewrite Data Flash by using the FDL.

3.1 FDL Sections

The functions, constants, and variables used in the FDL are allocated to the specified sections.
The table below lists the sections defined for the FDL.

Table 3-1 Sections Available for the FDL

Section name	Description
R_FDL_Text	Code section for the FDL. Contains the FDL programs.
R_FDL_Const	Constant data section for the FDL. Contains the constant data used inside the FDL.
R_FDL_Data	Variable data section for the FDL. Contains the variable data used inside the FDL.

3.2 FDL Resources

The table below lists the resources available for the FDL.

<R> Table 3-2 Resources Available for the FDL (Target: GHS-Version FDL V1.00)

Resource name	Size in bytes
R_FDL_Text section	3978
R_FDL_Const section	20
R_FDL_Data section	68
Maximum FDL stack size	Note

Note: Please contact our distributors or sales representatives.

3.3 Notes on Software Environment

Below are notes on the software environment:

Table 3-3 Notes on Software Environment

No.	Note
1	Reserve one stack area for the user and another stack area of the size specified in table 3-2 Resources Available for the FDL.
2	Do not destroy the FDL resources listed in table 3-2, Resources Available for the FDL.

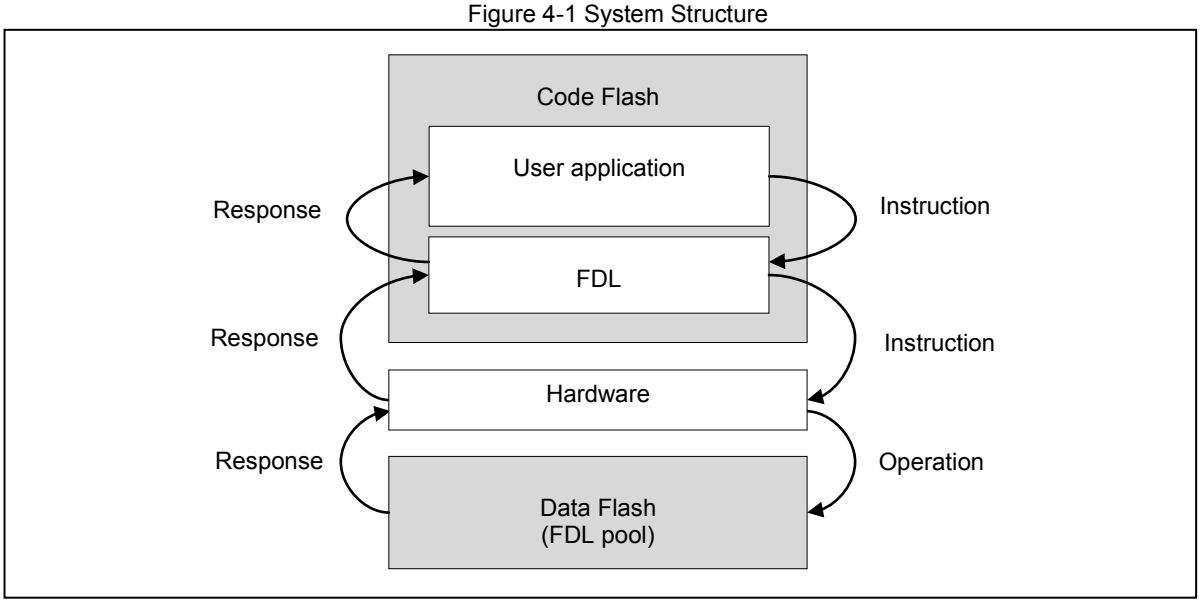
Chapter 4 FDL Architecture

This chapter describes the FDL architecture required for the user to rewrite Data Flash by using the FDL.

4.1 System Structure

The FDL offers interface for accessing the data flash area FDL pool defined by the user. The FDL is controlled via the hardware that operates the data flash.The arrows shown in the figure below indicate the flow of processing.

Figure 4-1 System Structure



4.1.1 FDL Pool

The FDL pool is a user-defined Data Flash area that is accessible by the FDL. The user program can access the Data Flash only by using this FDL pool in the Data Flash via the FDL.[Note]

The FDL pool size must be specified with the number of blocks in the Data Flash of the target device. For the procedure to specify the number of blocks, see section 6.1, Runtime Settings.

Note: Data can be directly read from the Data Flash via the user program without using the FDL.

4.2 Notes on FDL Architecture

Below are notes on the FDL architecture.

Table 4-1 Notes on FDL Architecture

No.	Note
1	The Data Flash is manipulated via the FDL functions, and it cannot be manipulated by the user directly. However, data can be read directly by the user program without using the FDL functions. The Data Flash cannot be read while a command of the R_FDL_Execute function is being executed. Before reading from the Data Flash, execute the R_FDL_Handler function to check that the command execution has been completed. Confirm the completion of command execution by referring to "status_enu" of the request structure [note].
2	The FDL can manipulate only Data Flash. It cannot rewrite Code Flash.
3	Do not stop the supply of input clock to the X1 pin (for connection to the main clock oscillator) during Data Flash operation. Data Flash operation needs the input clock supplied to the X1 pin.
4	The device is allowed to enter a power save mode only in the FDL standby state during Data Flash operation. To enable a power save mode in the FDL standby state, set the FDL standby state and then set the device to the HALT mode. The only power save mode that can be set in the FDL standby state is the HALT mode.
5	The FDL and FCL should not run at the same time.

Note: For details of the request structure, see section 6.2, Request Structure Settings.

Chapter 5 FDL Function

This chapter describes the functional specifications of the FDL required for the user to rewrite Data Flash by using the FDL.

5.1 FDL Functions / Commands of the R_FDL_Execute Function

The table below summarizes the FDL functions offered by the FDL.

Table 5-1 FDL Functions

FDL function name	Functional overview
R_FDL_Init	Initializes the FDL.
R_FDL_Execute	Starts execution of each command. Commands: R_FDL_CMD_ERASE, R_FDL_CMD_WRITE , R_FDL_CMD_BLANKCHECK, R_FDL_CMD_READ
R_FDL_Handler	Controls the FDL while it is running.
R_FDL_SuspendRequest	Requests the FDL to suspend.
R_FDL_ResumeRequest	Requests the FDL to return from suspended state.
R_FDL_StandBy	Sets the FDL to standby state.
R_FDL_WakeUp	Returns the FDL from standby state.
R_FDL_GetVersionString	Obtains FDL version information.

5.2 List of Commands

With the R_FDL_Execute function, the following commands can be executed.

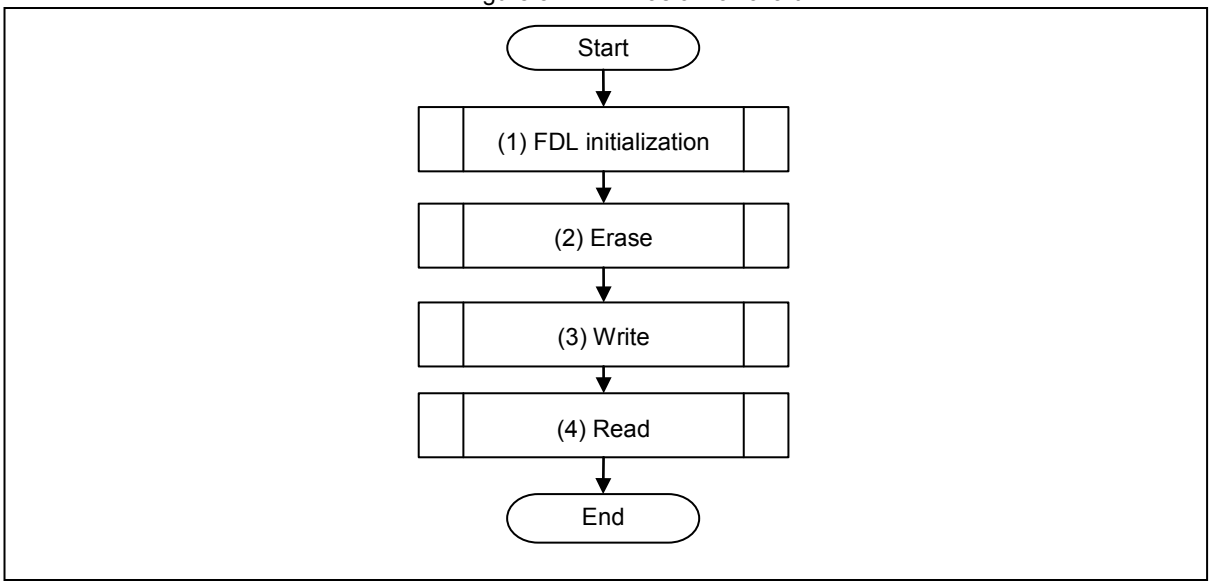
Table 5-2 Command

Command name	Functional overview
R_FDL_CMD_WRITE Command [Write]	This command reads data from the specified RAM area and writes the specified amount of data into the area starting with the specified address. The write area must be erased by executing the R_FDL_CMD_ERASE command in advance.
R_FDL_CMD_ERASE Command [Erase]	This command erases the content of the specified Data Flash blocks.
R_FDL_CMD_BLANKCHECK Command [Blank Check]	The blank check command of this FDL is used to check whether the target area is blank prior to read operation. If a device supporting this FDL performs read operation for a blank block containing no data, an undefined value is read out. Perform the blank check before reading to check whether the block contains data or not. This prevents an unwritten block from being treated as a data-written block. Use this command as necessary.
R_FDL_CMD_READ Command [Read]	This command reads the specified amount of data from the area starting with the specified address and stores the data in the specified RAM area. Before reading, be sure to check that the area contains data.

5.3 Basic Flowchart

The figure below shows the basic procedure to perform erase, write, and other operations for Data Flash by using the FDL.

Figure 5-1 FDL Basic Flowchart



(1) FDL initialization

Execute the R_FDL_Init function to initialize the FDL execution environment. The R_FDL_Init function must be executed only once[Note1] to initialize the FDL when the user system is started up. Interrupt should be disabled in certain periods during execution of the R_FDL_Init function. Be sure to use the NP and ID of the PSW register to disable interrupts. For details about interrupt control during execution of the R_FDL_Init function, see the description of the R_FDL_Init function in section 6.5, FDL Functions.

Note1: See table 5-3, Notes on the Functional Specifications of the FDL.

(2) Erase

First, execute the R_FDL_Execute function (using the R_FDL_CMD_ERASE command).Then, execute the R_FDL_Handler function. By repeating its execution, continue the erase operation, and finally confirm the termination.

(3) Write

First, execute the R_FDL_Execute function (using the R_FDL_CMD_WRITE command).Then, execute the R_FDL_Handler function. By repeating its execution, continue the write operation, and finally confirm the termination.

(4) Read[Note2]

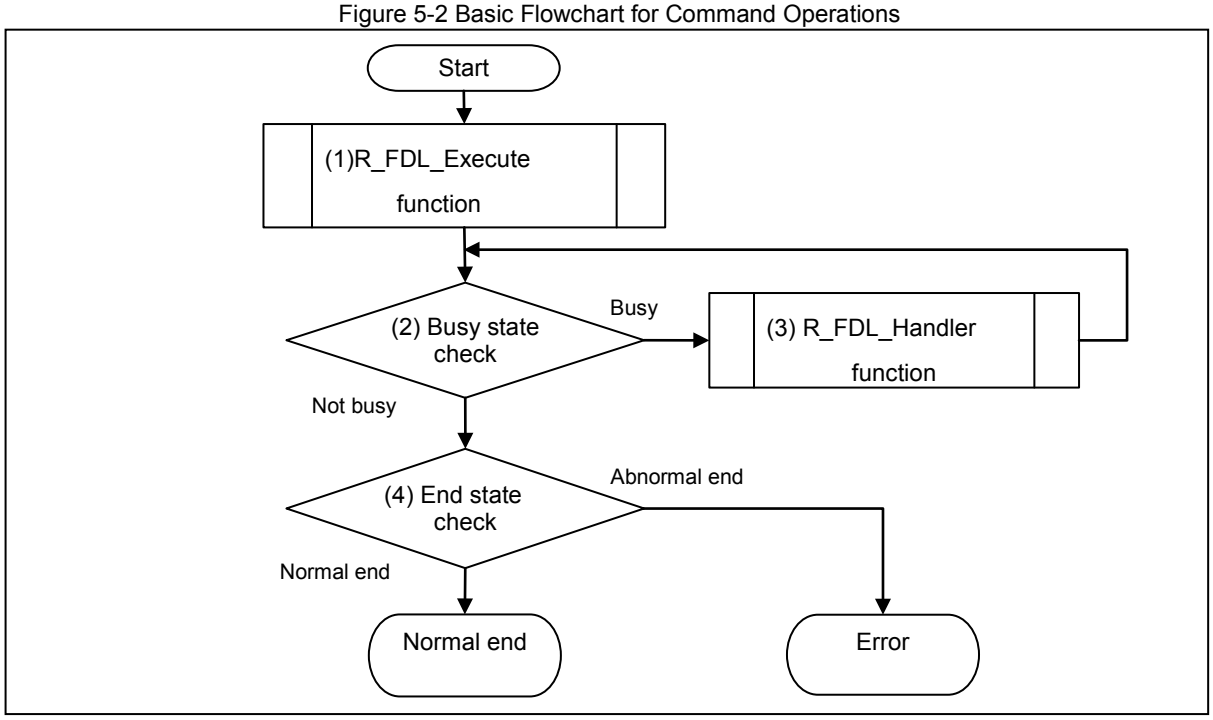
Execute the R_FDL_Execute function (the R_FDL_CMD_READ command). Execution of the R_FDL_CMD_READ command does not need execution of the R_FDL_Handler function.

Note 2: See table 4-1, Notes on FDL Architecture.

5.4 Flowchart for Command Operations

The figure below shows the basic procedure to perform erase and write operations for Data Flash by using the FDL.

Figure 5-2 Basic Flowchart for Command Operations



(1) R_FDL_Execute function

Perform operations for Command.

(2) Busy state check

Check status_enu of the request structure [Note]. Repeat the execution of the R_FDL_Handler function until a value other than R_FDL_BUSY is returned.

Note: For details of the request structure, see section 6.2, Request Structure Settings.

(3) R_FDL_Handler function

Control the FDL while it is running. By repeating the execution of the R_FDL_Handler function, continue the Command operation.

(4) End state check

If status_enu is R_FDL_OK, the operation should terminate normally. Otherwise, it should terminate with an error.

5. 4. 1 During Command Execution/During Busy

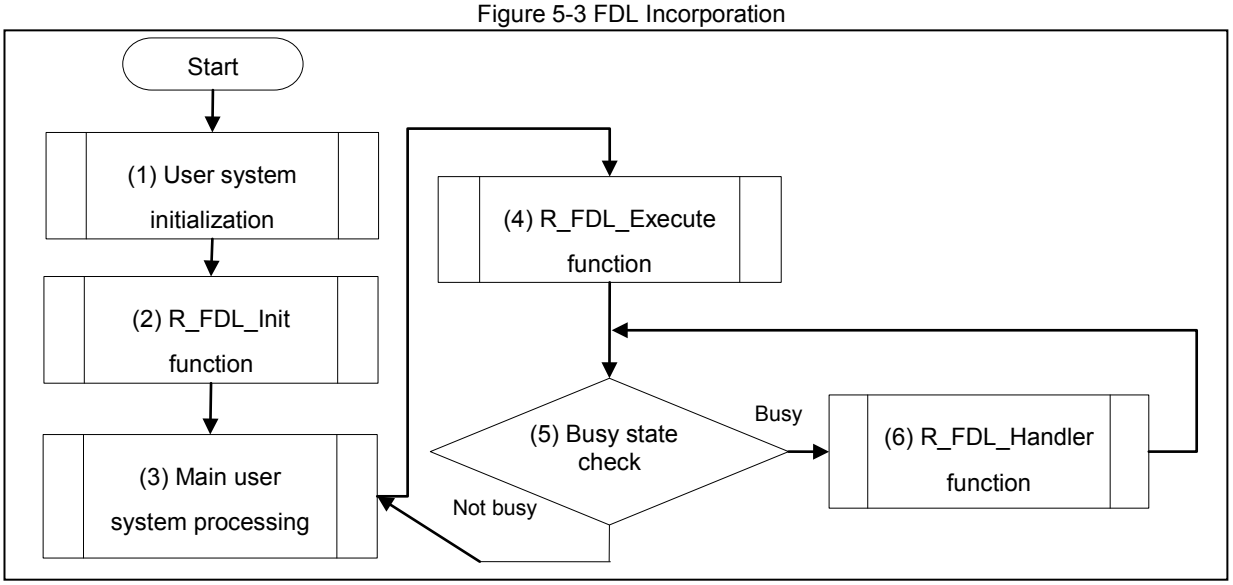
The time period from the execution of the (1) R_FDL_Execute function (busy) until the device leaves the busy state after the (3) R_FDL_Handler function is executed is called “during command execution” or “during busy” in this user’s manual. See the time period of command execution shown in figure 5-4, Example of Command Operation Sequence.

5.5 FDL Incorporation into the User System

Figure 5-3 shows an example of the procedure for incorporating the FDL into the user system. After executing the R_FDL_Init function once at the time of initialization, execute the R_FDL_Execute function[Note] and the R_FDL_Handler function according to the desired timing in the main part of the user program.

Note: For the use of the R_FDL_CMD_ERASE, R_FDL_CMD_WRITE, or R_FDL_CMD_BLANKCHECK

Figure 5-3 FDL Incorporation



(1) User system initialization

Startup routine for user system initialization.

(2) R_FDL_Init function

Initialize the FDL execution environment. Execute the R_FDL_Init function only once when starting the user system. Note: See table 5-3, Notes on the Functional Specifications of the FDL.

(3) Main user system processing

Main processing by the user system.

(4) R_FDL_Execute function

Execute Command operations.

(5) Busy state check

Check status_enu of the request structure [Note]. Repeat the execution of the R_FDL_Handler function until R_FDL_BUSY is changed to a different value.

Note: For details of the request structure, see section 6.2, Request Structure Settings.

(6) R_FDL_Handler function

Control the FDL while it is running. By repeating the execution of the R_FDL_Handler function, continue the Data Flash operation.

5.6 Back Ground Operation(BGO)

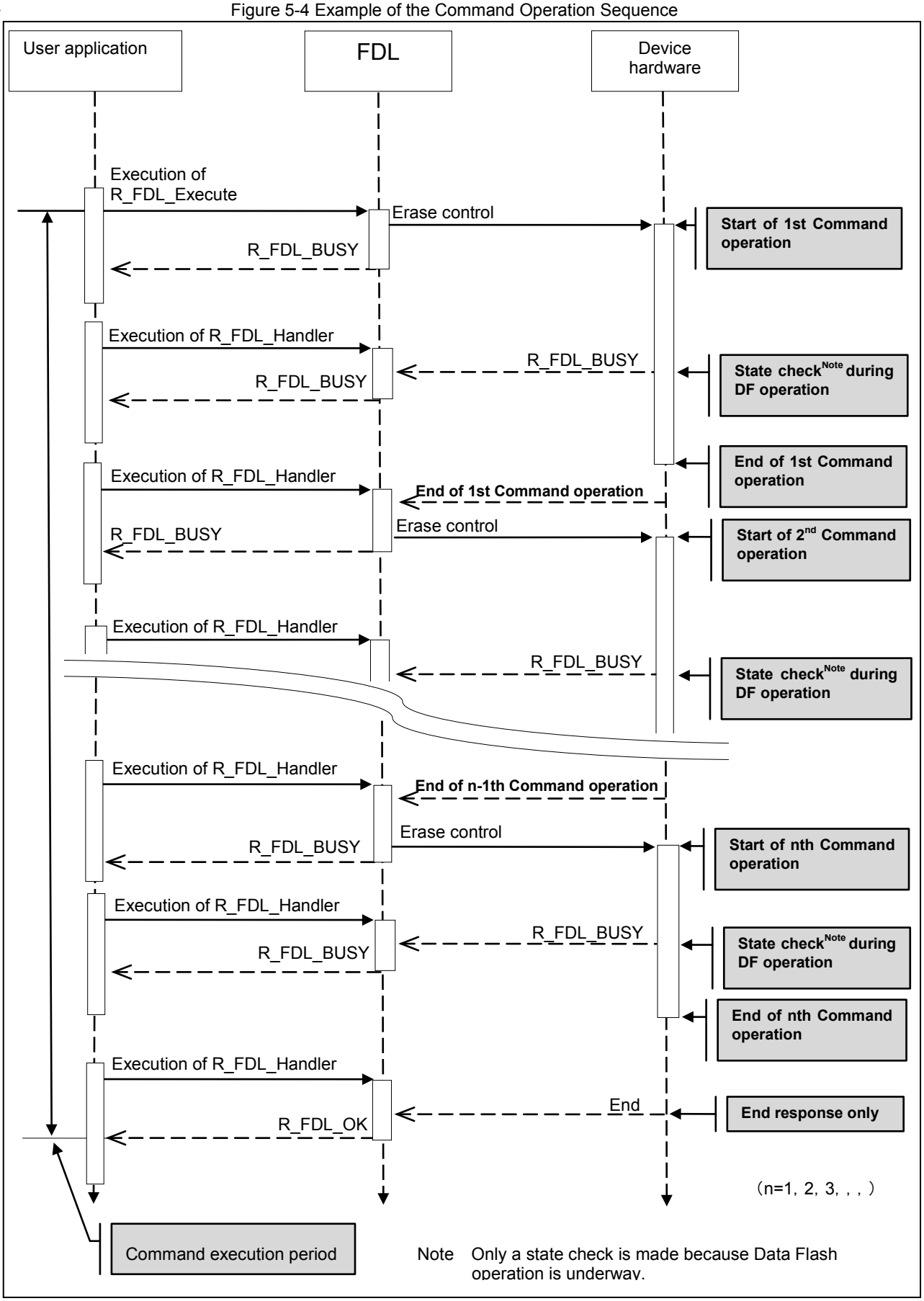
The R_FDL_Execute function is used for erase, write, blank check operations for Data Flash. This function requests the device hardware to perform required operation and then immediately returns control to the user application. This allows the user application to run during the Data Flash operation. This function is called BGO (Back Ground Operation).

In some cases, the device hardware operation might be divided into multiple operations, each performed on a separate occasion. The first operation is conducted by calling the R_FDL_Execute function. The second and subsequent operations are triggered by calling the R_FDL_Handler function. Thus, there is a need to call the R_FDL_Handler function multiple times. Processing is suspended from the time each separate operation is completed until the next one is triggered. Therefore, as the time interval between R_FDL_Handler function calls increases, so too does the overall processing time.

To determine whether or not the operation requested from the R_FDL_Execute function has normally ended, you should call the R_FDL_Handler function from the user application and check the device hardware operation.

The figure below gives an example of the hardware operation sequence. No more operations take place after the nth operation. A similar sequence is implemented with a command other than the R_FDL_CMD_WRITE command.

<R> Figure 5-4 Example of the Command Operation Sequence



5.7 Suspend and Resume

[Suspend]

The suspend feature enables you to stop ongoing erase or write operation. The state of the FDL inactivated by this feature is called FDL suspended state. Note that you can newly write data to memory while the FDL is in suspended state. The suspend feature is available only to stop erase or write operation. If you attempt to use this feature to stop other operations, status_enu of the request structure Note temporarily becomes R_FDL_SUSPENDED.

Note: For details of the request structure, see section 6.2, Request Structure Settings.

[Resume]

The resume feature returns the FDL from suspended state, making it possible to resume the suspended erase or write operation. This return from the FDL suspended state is called FDL resume.

[Timing]

After the R_FDL_SuspendRequest function is executed, a wait occurs upon one hardware operation (the device's operation is divided into several hardware operations). That is, after that function is executed, a delay is introduced before a transition to FDL suspended state. See the figure below.

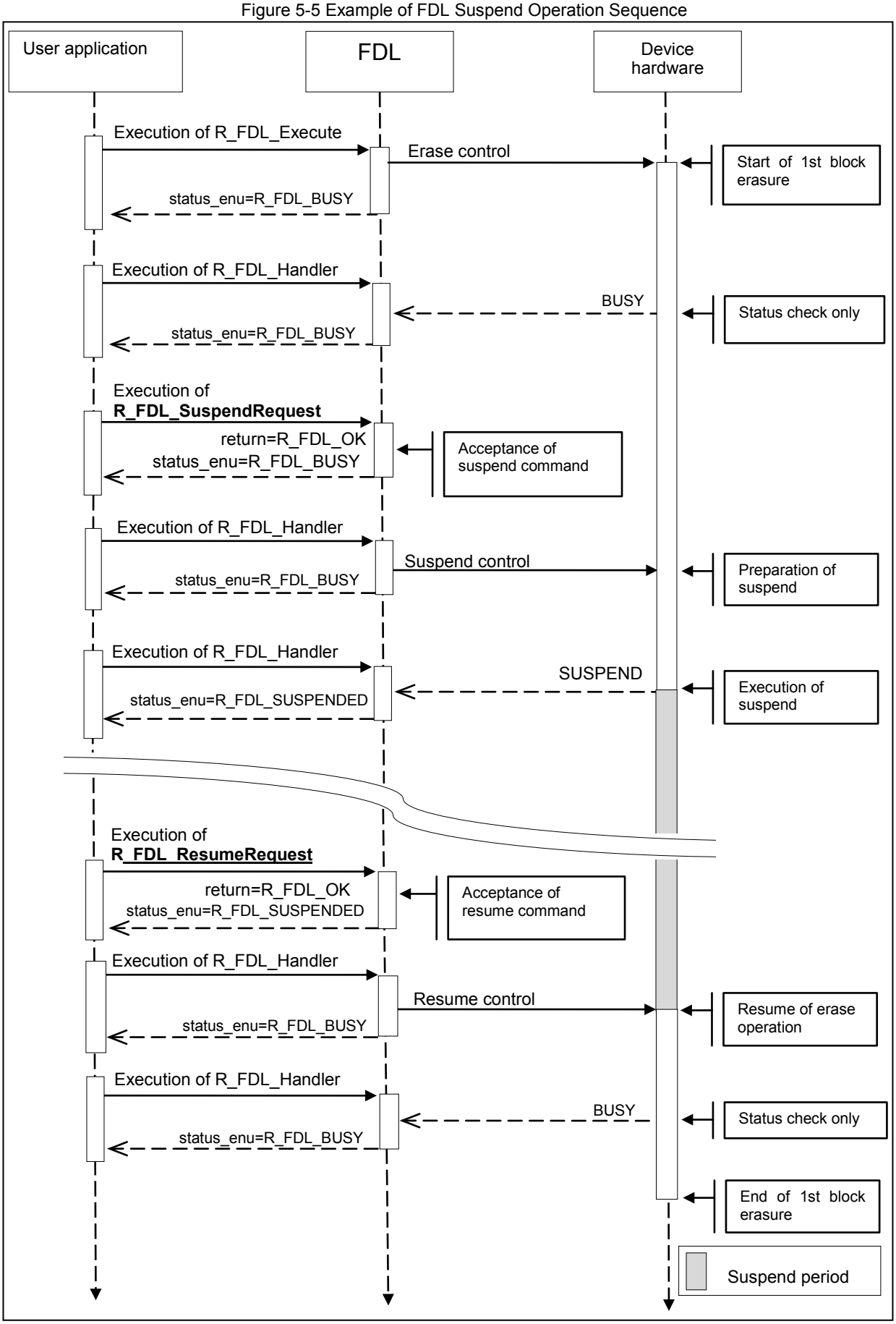
[Example of use]

<R> If data needs to be written for emergency, the erase operation currently in progress can be suspended to write that data.

[Flowcharts]

See the description of R_FDL_SuspendRequest and R_FDL_ResumeRequest in section 6.5, FDL Functions.

Figure 5-5 Example of FDL Suspend Operation Sequence



5.8 Standby and Wakeup

[Standby]

The standby feature enables you to stop ongoing erase and write operations. The state of the FDL inactivated by this feature is called FDL standby state. The FDL can be used while it is in FDL suspended state. But, it cannot be used while it is in FDL standby state.

[Wakeup]

The wakeup feature returns the FDL from FDL standby state, making it possible to resume the suspended erase or write operation.

[Timing]

After the R_FDL_StandBy function is executed, the timing feature forcibly aborts the ongoing hardware operation without waiting for the end of that operation (the device's operation is divided into several hardware operations).

[Example of Use]

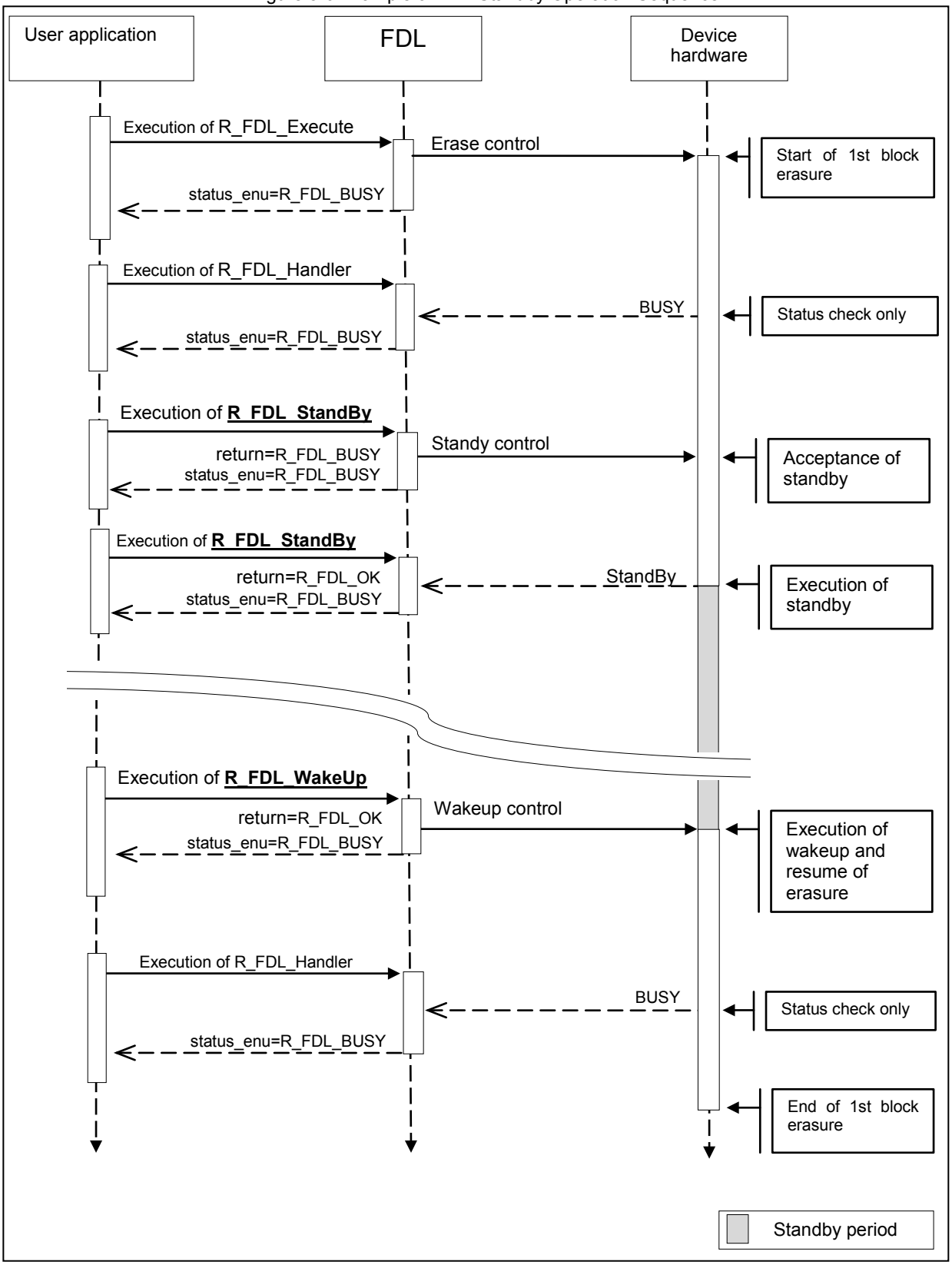
The standby and wakeup features are available to set the device to power save mode immediately during Data Flash operation. Note that you cannot set the device to power safe mode during Data Flash operation.

Note: Enabled in the HALT mode only. See table 4-1, Notes on FDL Architecture.

[Flowcharts]

See the description of R_FDL_StandBy and R_FDL_WakeUp in section 6.5, FDL Functions.

Figure 5-6 Example of FDL Standby Operation Sequence



<R> 5.9 Notes on the Functional Specifications of the FDL

Below are notes on the functional specifications of the FDL.

Table 5-3 Notes on the Functional Specifications of the FDL(1/2)

No.	Note
1	Timeout processing is not performed in the FDL. When timeout processing of an FDL function is required, use the watchdog timer or the like in the user program to perform the timeout processing.
2	If a reset or power-down occurs during write or erase operation, the operation is interrupted and the contents of the block being written or erased become undefined. Before using the block whose contents became undefined, perform the erase operation again.
3	The R_FDL_Init function must be executed only once to initialize the FDL when the user system is started up. However, in the following case, the R_FDL_Init function can be executed again. -- [Re-execution of the R_FDL_Init function] -- In the write or erase operation, if the FDL is in the R_FDL_BUSY state even after the timeout period has elapsed (see chapter 7, FDL Processing Time), execute the R_FDL_Init function to forcibly stop the ongoing write or erase operation.To perform Data Flash operation after timeout, follow the basic procedure again starting with the R_FDL_Init function (see figure 5-1, FDL Basic Flowchart). Thus,Initialize the request structure by the user. R_FDL_BUSY: status_enu of the request structure
4	When this FDL is used with a device, the erased area of Data Flash in that target device contains an undefined value. Thus, you get an undefined value when reading from the erased area. To prevent a read operation from being mistakenly conducted, supervise write operations to determine the areas to which data has been written. Note that you can detect blank state by using a blank check. Thus, perform a blank check as necessary.
5	Before writing data to an area, be sure to erase this area.
6	You cannot operate the FDL suspend or FDL standby feature concurrently in multiple operations.
7	The FDL does not support multitask execution. (1)Do not execute the FDL functions during interrupt processing. (2) Do not execute the FDL functions from multiple tasks in the OS.
8	If you suspend an erase operation and then resume it, the erase operation is performed again. In this case, however, the number of erase operations does not increase.
9	If the suspend feature is not effective for a FDL function, a suspend request is ignored while that function is being executed.
10	Once the FDL is set to suspended state, it is terminated from suspended state after write operation. Some commands such as “blank check” do not support the suspend feature. Even when these commands are executed, the library returns FDL suspended state.

Note: For details of the request structure, see section 6.2, Request Structure Settings.

Table 5-4 Notes on the Functional Specifications of the FDL(2/2)

No.	Note
11	The suspend feature disables the following sequences. <<Disabled sequences>> ・R_FDL_CMD_ERASE command --> FDL suspended state --> R_FDL_CMD_ERASE command ・R_FDL_CMD_WRITE command --> FDL suspended state --> R_FDL_CMD_ERASE command ・R_FDL_CMD_WRITE command --> FDL suspended state --> R_FDL_CMD_WRITE command Only the following sequence is enabled. <<Enabled sequence>> ・R_FDL_CMD_ERASE command --> FDL suspended state --> R_FDL_CMD_WRITE command (However, the R_FDL_CMD_WRITE command is enabled only for blocks other than the block that is being erased.)
12	While the FDL is in standby state, it cannot be used.
13	Interrupts are disabled during the execution of the R_FDL_Init function. For details about interrupt control during this execution, see the description of the R_FDL_Init function in section 6.5, FDL Functions.

Chapter 6 User Interface

This chapter describes the user interface offered by the FDL.

6.1 Runtime Settings

The fdl_descriptor.h file contains the runtime settings required for using the FDL. Some of the settings below might vary depending on the user. So, check the settings below before configuring them. The other settings are reserved for the system and thus cannot be changed.

(1) `#define FDL_CPU_FREQUENCY_MHZ 80`

Set the CPU frequency. The frequency used inside the FDL is calculated from the set CPU frequency.

Note: See notes table 2-2, Notes on Hardware Environment.

(2)
```
#define AUTHENTICATION_ID      { 0xFFFFFFFF, ¥
                                 0xFFFFFFFF, ¥
                                 0xFFFFFFFF, ¥
                                 0xFFFFFFFF }
```

Set the value of an authentication ID. For details of ID authentication, see the user's manual for the target device.

(3) `#define EEL_VIRTUALBLOCKSIZE 0u`

This value is reserved for the system. Because it should be fixed at 0u, do not change it.

(4) `#define FDL_POOL_SIZE 512u`

Set the size of the FDL pool area. Specify the number of blocks of Data Flash installed on the target device.
The FDL recognizes a Data Flash area by the set size.

Example) When the size of Data Flash is 32 Kbytes, FDL_POOL_SIZE is 512.

(5)
```
#define EEL_POOL_START  0u
#define EEL_POOL_SIZE   0u
```

Set the values to 0u. These settings are reserved for the system. Be sure to set them to 0u.

6.1.1 Note on Runtime Settings

Table 6-1 lists a note on the runtime settings.

Table 6-1 Note on Runtime Settings

No.	Note
1	Be sure to check the values of the following definitions in the file “fdl_descriptor.h”. **#define CPU_FREQUENCY_MHZ** **#define AUTHENTICATION_ID** **#define EEL_VIRTUALBLOCKSIZE** **#define FDL_POOL_SIZE** **#define FDL_EEL_POOL_START** **#define FDL_EEL_POOL_SIZE** The EEL_POOL_START and EEL_POOL_SIZE definitions are reserved for the system. Be sure to set "0u" for them. All initial values of ID authentication are 0xFF.

6.2 Request Structure Settings

The definitions in the figure below are called request structure. The request structure is described in the file “r_fdl_types.h”.

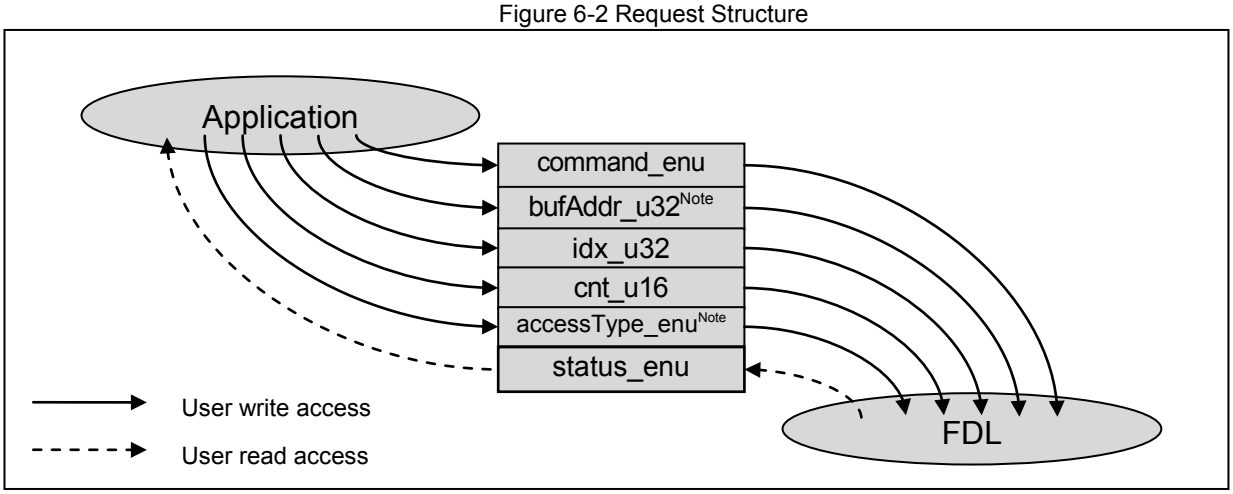
Figure 6-1 Definition of the request structure

```
typedef volatile struct R_FDL_REQUEST_T   {
    uint32_t        command_enu;
    uint32_t        bufAddr_u32;
    uint32_t        idx_u32;
    uint16_t        cnt_u16;
    uint32_t        accessType_enu;
    uint32_t        status_enu;
} r_fdl_request_t;
```

Commands are executed by a single function. Commands and data sizes are then passed to the FDL via the request structure. The FDL state and error information are obtained via the request structure.

In subsequent sections, write access to the request structure from the user is called “user write access”, and read access to it from the user is called “user read access”.

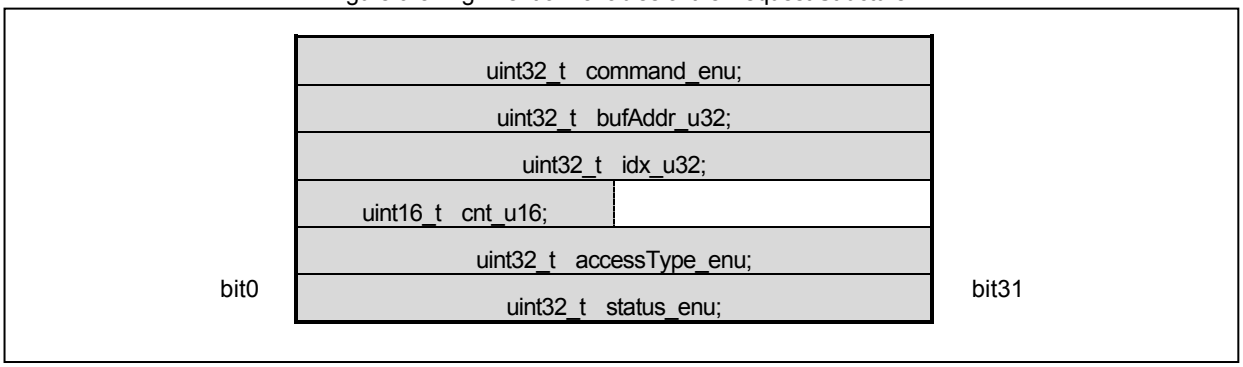
Figure 6-2 Request Structure



Note: bufAddr_u32 and accessType_enu may be written from the FDL. User read access is also possible. For more information about how to do this, see the description of the R_FDL_Execute function in section 6.5, FDL Functions.

The request structure should not be changed by the user except for adding dummy. For details, see table 6-2, Notes on Request Structure Settings.

Figure 6-3 Alignment of Variables of the Request Structure



6.2.1 User Write Access

The following outlines settings for each member of the request structure at user write access.

(1) command_enu

Commands to be set in the common executable function.

Command name (macro name)	Description
R_FDL_CMD_ERASE	Erase
R_FDL_CMD_WRITE	Write
R_FDL_CMD_BLANKCHECK	Blank check
R_FDL_CMD_READ	Read

(2) bufAddr_u32

Specify the start address of the data buffer used for R_FDL_CMD_WRITE, R_FDL_CMD_READ command execution.

Associated command (macro name)	Setting
R_FDL_CMD_WRITE	Start address of the data buffer[Note1]
R_FDL_CMD_READ	Start address of the data buffer[Note2]

Note 1: Buffer which contains data written by the user

Note 2: Buffer which stores data read from the Data Flash

(3) idx_u32

Specify the index used for each command. For more information about how to do this, see the description of the R_FDL_Execute function in section 6.5, FDL Functions.

Associated command name (macro name)	Setting
R_FDL_CMD_ERASE	Block number for initiating erase
R_FDL_CMD_WRITE	Start address for performing write
R_FDL_CMD_BLANKCHECK	Start address for performing a blank check
R_FDL_CMD_READ	Start address for performing read

(4) cnt_u32

Specify the number used for each command. For more information about how to do this, see the description of the R_FDL_Execute function in section 6.5, FDL Functions.

Associated command name (macro name)	Setting
R_FDL_CMD_ERASE	Amount of memory to be erased. Expressed in units of blocks.
R_FDL_CMD_WRITE	Amount of data to be written. Expressed in 4-byte units.
R_FDL_CMD_BLANKCHECK	Amount of data which should be subjected to a blank check. Expressed in 4-byte units.
R_FDL_CMD_READ	Amount of data to be read. Expressed in 4-byte units.

(5) accessType_enu

Right to access Data Flash. Set R_FDL_ACCESS_USER.

Macro name	Description
FDL_ACCESS_USER	Enables access to the user pool from the FDL.

Remarks

Macro name	Description
FDL_ACCESS_NONE	Should not be set by the user
FDL_ACCESS_EEL	Setting reserved for the system. Should not be set by the user.

6.2.2 User Read Access

- status_enu

FDL state and error information. For details, see section 6.5, FDL Functions. status_enu is also referenced when the R_FDL_SuspendRequest or R_FDL_ResumeRequest function is executed (as in the case of R_FDL_Execute or R_FDL_Handler).

6.2.3 Note on Request Structure Settings

Table 6-2 lists a note on the request structure settings.

Table 6-2 Note on Request Structure Settings

No.	Note
1	Do not change the position of each variable of the request structure. Variables must be aligned as shown in figure 6-3, Alignment of Variables of the Request Structure . Otherwise, the interface becomes incompatible with the FDL. Example: When the structure packing "-Xpack=1" of the CC-RH compiler options is used, "accessType_enu" and "status_enu" become out of alignment by 16 bits. To avoid this problem, insert dummy data as shown below: typedef volatile struct R_FDL_REQUEST_T { -- omitted -- uint16_t cnt_u16; uint16_t dummy;←Add this. r_fdl_accessType_t accessType_enu; -- omitted --
2	Every time before executing a command, check the variables of the request structure and assign values.
3	The request structure is accessed by the FDL while command execution is in progress. Do not destroy the request structure during command execution.
4	If the R_FDL_Handler function is executed again after an error has occurred in status_enu of the request structure, the FDL initializes the status_enu value to R_FDL_OK.

6.3 FDL Function Calls

This section describes how to call the FDL functions from a user program written in C or assembly language.

- C language

When a FDL function is called from a user program in C language in the same way as a normal C function is called, the FDL function's parameters are passed to the FDL as arguments and the required processing is performed.

- Assembly language

Before a FDL function is called from a user program in assembly language, processing (such as parameter setup and return address setup) is performed according to the function calling conventions for the C compiler package which is used as development environment by the user. Then, the function is called using a jarl instruction. As a result, the FDL function's parameters are passed to the FDL as arguments and the required processing is performed.

Remarks

1. To call the FDL functions offered by the FDL from a user program, you should define the following standard header file and include it in that program:
r_fdl. h: Standard header file

2. If a FDL function other than the R_FDL_Init function is called before calling the R_FDL_Init function, proper operation is not guaranteed.

6.4 Data Types

Below are the data types of the parameters to be specified for calling the FDL functions offered by the FDL.

Macro name	Description
uint8_t	Unsigned 8-bit integers (unsigned char)
uint16_t	Unsigned 16-bit integers (unsigned short)
uint32_t	Unsigned 32-bit integers (unsigned long)

6.5 FDL Functions

The subsequent sections describe the FDL functions offered by the FDL. These functions appear in the following format:

Name

Description

C language format

Pre-conditions

Post-conditions

Parameters

Parameter	Description

Return values

Value	Macro name	Description

Commands

Value	Macro name	Description

Reference

Member	Value	Macro name

Reference information such as the request structure

Settings

Member	Value	Macro name

Settings information such as the request structure

Others

Additional descriptions such as flowcharts

R_FDL_Init

Description

This function initializes the FDL execution environment. More specifically, it initializes the following:

- Flash hardware initialization
- Confirmation of runtime settings (CPU operating frequency, ID authentication, FDL pool size, etc.)
- FDL internal variables

Interrupts are disabled during the execution of the R_FDL_Init. For details, see "Others" on the following page.

C language format

```
r_fdl_status_t R_FDL_Init( const r_fdl_descriptor_t *descriptor_pstr)
```

Pre-conditions

When the user program is started, the R_FDL_Init function must be executed only once [Note] to initialize the FDL and perform ID authentication.

Note: See table 5-3, Notes on the Functional Specifications of the FDL.

Post-conditions

None

Parameters

Parameter	Description
const r_fdl_descriptor_t *descriptor_pstr	Runtime settings pointer

Return Values

This function can return the values below.

Macro name	Value	Category	Description
R_FDL_OK	0x00	Meaning	Processing normally ended.
		Cause	Normal operation.
		Action to be taken	None
R_FDL_ERR_ CONFIGURATION	0x03	Meaning	Execution of this R_FDL_Init function was rejected.
		Cause	Any runtime settings (excluding ID authentication) or the argument pointer is incorrect.
		Action to be taken	Set correct values for runtime settings or the argument pointer.
R_FDL_ERR_ PROTECTION	0x05	Meaning	Execution of this R_FDL_Init function was rejected.
		Cause	1) The ID authentication value in runtime settings is incorrect. [Note] 2) An access to the data flash was rejected due to a hardware fault.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.
R_FDL_ERR_ INTERNAL	0x0D	Meaning	Execution of this R_FDL_Init function failed.
		Cause	An unexpected error that cannot be identified by the FDL 1) The RAM used by the FDL may have been destroyed by the user program. 2) The FDL may have been destroyed.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.

Note: Be sure to set a correct ID also for devices that do not require ID authentication for the data flash.

Commands
Reference
Settings

None

Others

[Disabling interrupts during execution of the R_FDL_Init function]

Interrupts should be disabled in certain periods during execution of the R_FDL_Init function. Be sure to use the NP and ID of the PSW register to disable interrupts. The FDL Type 01 provides an interface for callback functions to be called before and after the interrupt-disabled period in the R_FDL_Init function. Be sure to use either a) or b) method below to disable interrupts.

a) Disable interrupts before execution of the R_FDL_Init function and enable interrupts after execution of the R_FDL_Init function.
b) Disable interrupts within the R_FDL_User_CriticalSetionBegin function and enable interrupts within the R_FDL_User_CriticalSetionEnd function.

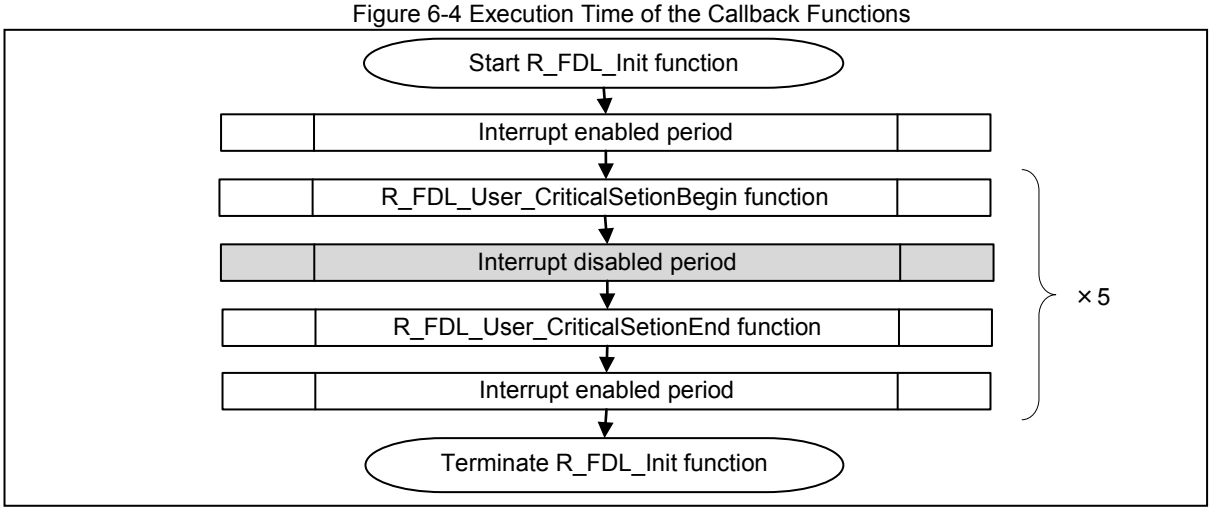
Since the processing in the provided callback function is a sample program, it needs to be tailored by the user. Change the sample program as required and be sure to evaluate it adequately.
If callback functions are unnecessary, delete the sample code of each callback function.

File name: r_fdl_user.c
Callback function names: R_FDL_User_CriticalSetionBegin, R_FDL_User_CriticalSetionEnd

- Callback function execution timing

The R_FDL_User_CriticalSetionBegin function is executed right before the interrupt-disabled period during execution of the R_FDL_Init function, and the R_FDL_User_CriticalSetionEnd function is executed right after the interrupt-disabled period.

Figure 6-4 Execution Time of the Callback Functions



R_FDL_Execute

Description

This function performs erase, write, blank check operations for Data Flash.

Commands, which are treated as parameters, are specified in the request structure (r_fdl_requst_t) for execution.

C language format

```
void R_FDL_Execute(r_fdl_request_t *request_pstr)
```

Pre-conditions

The R_FDL_Init function should execute and return R_FDL_OK.

Post-conditions

- After the R_FDL_Execute function normally ends, status_enu of the request structure is R_FDL_BUSY.
- Execute the R_FDL_Handler function repeatedly while status_enu is R_FDL_BUSY.

Parameters

Parameter	Description
r_fdl_request_t * request_pstr	Request structure pointer

Return values

None

Commands

This function can use command_enu below of the request structure.

Value	Macro name	Description
0x00	R_FDL_CMD_ERASE	Erase
0x01	R_FDL_CMD_WRITE	Write
0x02	R_FDL_CMD_BLANKCHECK	Blank check
0x03	R_FDL_CMD_READ	Read

Reference

The following table lists status_enu values of the request structure that are supported for all commands of R_FDL_Execute and R_FDL_Handler that follows.

[Values common to all commands]

Macro name	Value	Category	Description
R_FDL_OK	0x00	Meaning	Processing normally ended.
		Cause	Normal operation.
		Action to be taken	None
R_FDL_BUSY	0x01	Meaning	Command operation has started normally or the command is being executed.
		Cause	Normal operation.
		Action to be taken	Continue operation.
R_FDL_ERR_PARAMETER	0x04	Meaning	Execution of this command was rejected.
		Cause	An invalid parameter was specified.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.
R_FDL_ERR_PROTECTION[Note]	0x05	Meaning	The command to be executed was rejected.
		Cause	Parameters in the FDL or parameters that have been set in the data flash control hardware via the FDL were illegally modified during command execution.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.
R_FDL_ERR_REJECTED	0x06	Meaning	The command to be executed was rejected.
		Cause	Another command or FDL function is being executed.
		Action to be taken	Re-execute the command after the hardware operation is completed.
R_FDL_ERR_COMMAND	0x0A	Meaning	The command to be executed was rejected.
		Cause	An invalid command was specified.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.
R_FDL_ERR_INTERNAL	0x0D	Meaning	Execution of this command failed.
		Cause	An unexpected error that cannot be identified by the FDL 1) The RAM used by the FDL may have been destroyed by the user program. 2) The FDL may have been destroyed.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.

Note: In the case of R_FDL_CMD_READ command execution,this error does not occur.

[Value for R_FDL_CMD_READ command only]

status_enu	Value	Category	Description
R_FDL_ERR_ECC_SED	0x0B	Meaning	Data Flash operation was executed and the corrected data was read.
		Cause	An ECC error (1-bit error)Note was detected.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.
R_FDL_ERR_ECC_DED	0x0C	Meaning	Data Flash operation was executed and the data which failed to be corrected was read.
		Cause	An ECC error (2-bit error)Note was detected.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.

Note: For details of ECC, refer to the user's manual of the target device.

[Value for R_FDL_CMD_ERASE command only]

status_enu	Value	Category	Description
R_FDL_ERR_ERASE	0x08	Meaning	Erase failed due to a hardware problem.
		Cause	Data Flash might be faulty.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.

[Value for R_FDL_CMD_WRITE command only]

status_enu	Value	Category	Description
R_FDL_ERR_WRITE	0x07	Meaning	Write failed due to a hardware problem.
		Cause	1. The area to write data to might not be erased. 2. Data Flash might be faulty.
		Action to be taken	1. If the area to write data to is not erased, erase it and then re-execute the write. 2. If the area to write data to is already erased, use a bit check to investigate the cause of the problem.

[Value for R_FDL_CMD_BLANKCHECK command only]

status_enu	Value	Category	Description
R_FDL_ERR_BLANKCHECK	0x09	Meaning	The specified area is not blank.
		Cause	Data is already written.
		Action to be taken	1. If a blank check ensures the area is not blank, erase the area because it contains data. 2. If a blank check ensures the area is blank, perform the write operation.

[Value for R_FDL_SuspendRequest function , R_FDL_ResumeRequest function only]

status_enu	Value	Category	Description
R_FDL_SUSPENDED	0x02	Meaning	The FDL is being suspended.
		Cause	Normal operation
		Action to be taken	None

This function can assign the values below to bufAddr_u32 and FDL_accessType_t of the request structure.

Member	Description
bufAddr_u32	The address where the blank check resulted in error is obtained upon occurrence of R_FDL_ERR_BLANKCHECK.
accessType_t	FDL_ACCESS_NONE which indicates no access right is set at reset time.

Settings

This function uses the request structure settings below.

(1) Settings for R_FDL_CMD_ERASE Command Execution

Member	Description
command_enu	Set R_FDL_CMD_ERASE.
bufAddr_u32	Not used. Any settings are invalid. However, for safety design, treat invalid values as 0xFFFF_FFFF.
idx_u32	Set the start block number. Example 1) Block 0 Setting: 0 Example 1) Block 1 Setting: 1
cnt_u16	Set the number of blocks to be erased. Example 1) Erasing three blocks Setting: 3
accessType_enu	The access right is always R_FDL_ACCESS_USER.

(2) Settings for R_FDL_CMD_WRITE Command Execution

Member	Description
command_enu	Set R_FDL_CMD_WRITE.
bufAddr_u32	Start address of the data buffer[Note]. Example) Array for the data buffer with the name “data” bufAddr_u32=&data[0];
idx_u32	As the start address of the location to write data, specify a relative address. Set the first address of block 0 in the Data Flash to 0x0000_0000. Only 4-byte aligned addresses (whose lowest bytes are 0H, 4H, 8H, or CH) are specifiable, with the first address of block 0 in the Data Flash set to 0x0000_0000. Example 1) Writing data to the beginning of Data Flash block 0 Setting: 0x0000_0000 Example 2) Writing data to the first four bytes of Data Flash block 1 Setting: 0x0000_0044 Example 3) When the address shown below (the third address from the top of block 0 in the Data Flash) is specified before command execution, status_enu of the request structure becomes R_FDL_ERR_PARAMETER because the address is not aligned with a 4-byte boundary. Setting: 0x0000_0003
cnt_u16	4-byte units Example) Eight bytes Setting: 2
accessType_enu	The access right is always R_FDL_ACCESS_USER.

Note Buffer which contains data to be written by the user

(3) Settings for R_FDL_CMD_BLANKCHECK Command Execution

Member	Description
command_enu	Set R_FDL_CMD_BLANKCHECK.
bufAddr_u32	The address where the blank check resulted in error is obtained upon occurrence of R_FDL_ERR_BLANKCHECK. Since this member is used only when R_FDL_ERR_BLANKCHECK occurs, any settings are invalid. However, for safety design, treat invalid values as 0xFFFF_FFFF.
idx_u32	As the start address of the blank check, specify a relative address from the first address (0x0000_0000) of block 0 in the Data Flash. Only 4-byte aligned addresses (whose lowest bytes are 0H, 4H, 8H, or CH) are specifiable, with the first address of block 0 in the Data Flash set to 0x0000_0000. Example 1) Performing a blank check on the beginning of Data Flash block 0 Setting: 0x0000_0000 Example 2) Performing a blank check on the first 4 bytes of Data Flash block 1 Setting: 0x0000_0044 Example 3) When the address shown below (the third address from the top of block 0 in the Data Flash) is specified before command execution, status_enu of the request structure becomes R_FDL_ERR_PARAMETER because the address is not aligned with a 4-byte boundary. Setting: 0x0000_0003
cnt_u16	4-byte units Example) Eight bytes Setting: 2
accessType_enu	The access right is always R_FDL_ACCESS_USER.

(4) Settings for R_FDL_CMD_READ Command Execution

Member	Description
command_enu	Set R_FDL_CMD_READ
bufAddr_u32	Start address of the data buffer[Note]. Example) Array for the data buffer with the name “data” bufAddr_u32=&data[0]; Address where an error was detected when R_FDL_ERR_ECC_SED/R_FDL_ERR_ECC_DED occurred.
idx_u32	As the start address of read operation, specify a relative address from the first address (0x0000_0000) of block 0 in the Data Flash. Only 4-byte aligned addresses (whose lowest bytes are 0H, 4H, 8H, or CH) are specifiable, with the first address of block 0 in the Data Flash set to 0x0000_0000. Example 1) Reading data to the beginning of Data Flash block 0 Setting: 0x0000_0000 Example 2) Reading data to the first 4 bytes of Data Flash block 1 Setting: 0x0000_0044 Example 3) When the address shown below (the third address from the top of block 0 in the Data Flash) is specified before command execution, status_enu of the request structure becomes R_FDL_ERR_PARAMETER because the address is not aligned with a 4-byte boundary. Setting: 0x0000_0003
cnt_u16	4-byte units Example) Eight bytes Setting: 2
accessType_enu	The access right is always R_FDL_ACCESS_USER.

Note: Buffer which stores data read from the Data Flash

R_FDL_Handler

Description

This function continues command operations of the FDL and confirms their termination.

C language format

```
void R_FDL_Handler( void )
```

Pre-conditions

- The R_FDL_Init function should execute and return R_FDL_OK.
- The R_FDL_Execute function should execute, and status_enu of the request structure should be R_FDL_BUSY.

  Note: Execution of the R_FDL_CMD_READ command does not need execution of the R_FDL_Handler function.

Post-conditions

Repeat the execution of the R_FDL_Handler function until a value other than R_FDL_BUSY is returned.

Parameters
Return values
Commands

None

Reference

“Reference” of this function is described in combination with the R_FDL_Execute function. See “Reference” of the R_FDL_Execute function.

Settings

None

R_FDL_SuspendRequest

Description

This function requests the FDL to be suspended while it is running.

The ongoing write and erase operations are stopped. While the FDL is in suspended state, it permits further write operation.

C language format

```
r_fdl_status_t R_FDL_SuspendRequest( void )
```

Pre-conditions

- The R_FDL_Init function should execute and return R_FDL_OK.
- status_enu of the request structure should be R_FDL_BUSY for write or erase.
- The FDL should not be in the suspended state.

Post-conditions

The execution of the R_FDL_Handler function should be repeated until status_enu of the request structure is changed to R_FDL_SUSPENDED.

Parameters

None

Return values

This function can return the values below.

Macro name	Value	Category	Description
R_FDL_OK	0x00	Meaning	The FDL suspend request was successfully accepted.
		Cause	Normal operation.
		Action to be taken	None
R_FDL_ERR_REJECTED	0x06	Meaning	Execution of this R_FDL_SuspendRequest function was rejected.
		Cause	The FDL has not been initialized or is an invalid state.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.

Commands

None

Reference

This function can assign the value below to status_enu of the request structure.

status_enu	Value	Category	Description
R_FDL_BUSY	0x01	Meaning	During command execution.
		Cause	Normal operation.
		Action to be taken	Continue operation.

Settings

None

Others

“Flowchart” of this function is described in combination with the R_FDL_ResumeRequest function. See “Flowchart” of the R_FDL_ResumeRequest function.

R_FDL_ResumeRequest

Description

This function requests the FDL to return from suspended state.

C language format

```
r_fdl_status_t R_FDL_ResumeRequest( void )
```

Pre-conditions

- The R_FDL_Init function should execute and return R_FDL_OK.
- status_enu of the request structure should be R_FDL_SUSPENDED.

Post-conditions

The execution of the R_FDL_Handler function should be repeated until status_enu of the request structure is changed from R_FDL_SUSPENDED to a different value.

Parameters

None

Return values

This function can return the values below.

Macro name	Value	Category	Description
R_FDL_OK	0x00	Meaning	A request for return from the FDL suspended state was successfully accepted.
		Cause	Normal operation.
		Action to be taken	None
R_FDL_ERR_REJECTED	0x06	Meaning	Execution of this R_FDL_ResumeRequest function was rejected.
		Cause	The FDL has not been initialized or is an invalid state.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.

Commands

None

Reference

This function can assign the value below to statsu_enu of the request structure.
When the FDL returns from suspended status, status_enu of the request structure is changed back to the value which existed before the FDL went into suspended state.

status_enu	Value	Category	Description
R_FDL_SUSPENDED	0x02	Meaning	The FDL is suspended.
		Cause	Normal operation.
		Action to be taken	None

Settings

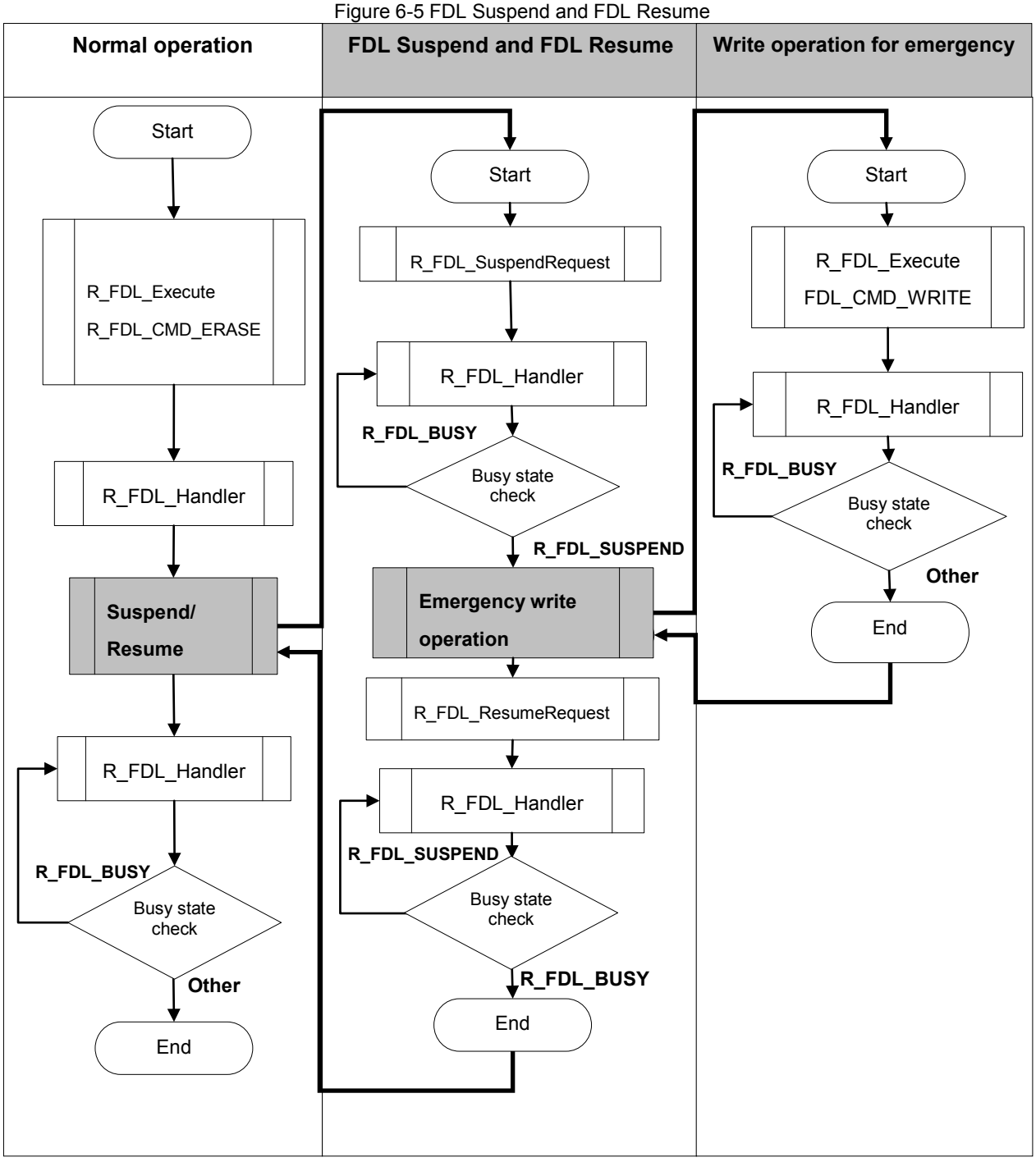
None

Others

[Flowchart]

The figure below shows the basic flow of execution of the R_FDL_SuspendRequest and FDL_ResumeRequst functions used to perform an example write operation for emergency.Error processing is omitted in this figure. Busy state is checked using status_enu of the request structure.

Figure 6-5 FDL Suspend and FDL Resume



Note :Error processing is omitted in this figure

R_FDL_StandBy

Description

This function sets the FDL to standby state while it is running. The Data Flash operation is stopped. While the FDL is in standby state, it does not permit any Data Flash operations.

C language format

```
r_fdl_status_t R_FDL_StandBy( void )
```

Pre-conditions

- The R_FDL_Init function should execute and return R_FDL_OK.
- status_enu of the request structure should have been set to R_FDL_BUSY by a write or erase operation.
- The FDL should not be in the standby state.

Post-conditions

- Execute the R_FDL_StandBy function repeatedly while the return value of the R_FDL_StandBy function is R_FDL_BUSY.
- Only the R_FDL_WakeUp function is executable after the FDL standby.

Parameters

None

Return values

This function can return the values below.

Macro name	Value	Category	Description
R_FDL_OK	0x00	Meaning	FDL standby started.
		Cause	Normal operation.
		Action to be taken	None
R_FDL_BUSY	0x01	Meaning	The FDL standby request was successfully accepted.
		Cause	Normal operation.
		Action to be taken	Continue operation.
R_FDL_ERR_REJECTED	0x06	Meaning	Execution of this R_FDL_StandBy function was rejected.
		Cause	The FDL has not been initialized or is an invalid state.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.

Commands

None

Reference

This function can assign the value below to status_enu of the request structure.

status_enu	Value	Category	Description
R_FDL_BUSY	0x01	Meaning	During command execution.
		Cause	Normal operation.
		Action to be taken	Continue operation.

Settings

None

Others

“Flowchart” of this function is described in combination with the R_FDL_WakeUp function. See “Flowchart” of the R_FDL_WakeUp function.

R_FDL_WakeUp

Description

This function returns the FDL from standby state.

C language format

```
r_fdl_status_t R_FDL_WakeUp( void )
```

Pre-conditions

- The R_FDL_Init function should execute and return R_FDL_OK.
- The return value of the R_FDL_StandBy function should have changed from R_FDL_BUSY to R_FDL_OK, indicating the FDL is in standby state.
- status_enu of the request structure should be R_FDL_BUSY.

Post-conditions

None

Parameters

None

Return values

This function can return the values below.

Macro name	Value	Category	Description
R_FDL_OK	0x00	Meaning	A request for return from the FDL standby state was successfully accepted.
		Cause	Normal operation.
		Action to be taken	None
R_FDL_ERR_REJECTED	0x06	Meaning	Execution of this R_FDL_WakeUp function was rejected.
		Cause	The FDL has not been initialized or is an invalid state.
		Action to be taken	Stop further Flash operation and investigate the cause of the problem.

Commands

None

Reference

This function can assign the value below to status_enu of the request structure.

status_enu	Value	Category	Description
R_FDL_BUSY	0x01	Meaning	During command execution.
		Cause	Normal operation.
		Action to be taken	Continue operation.

Settings

None

Others

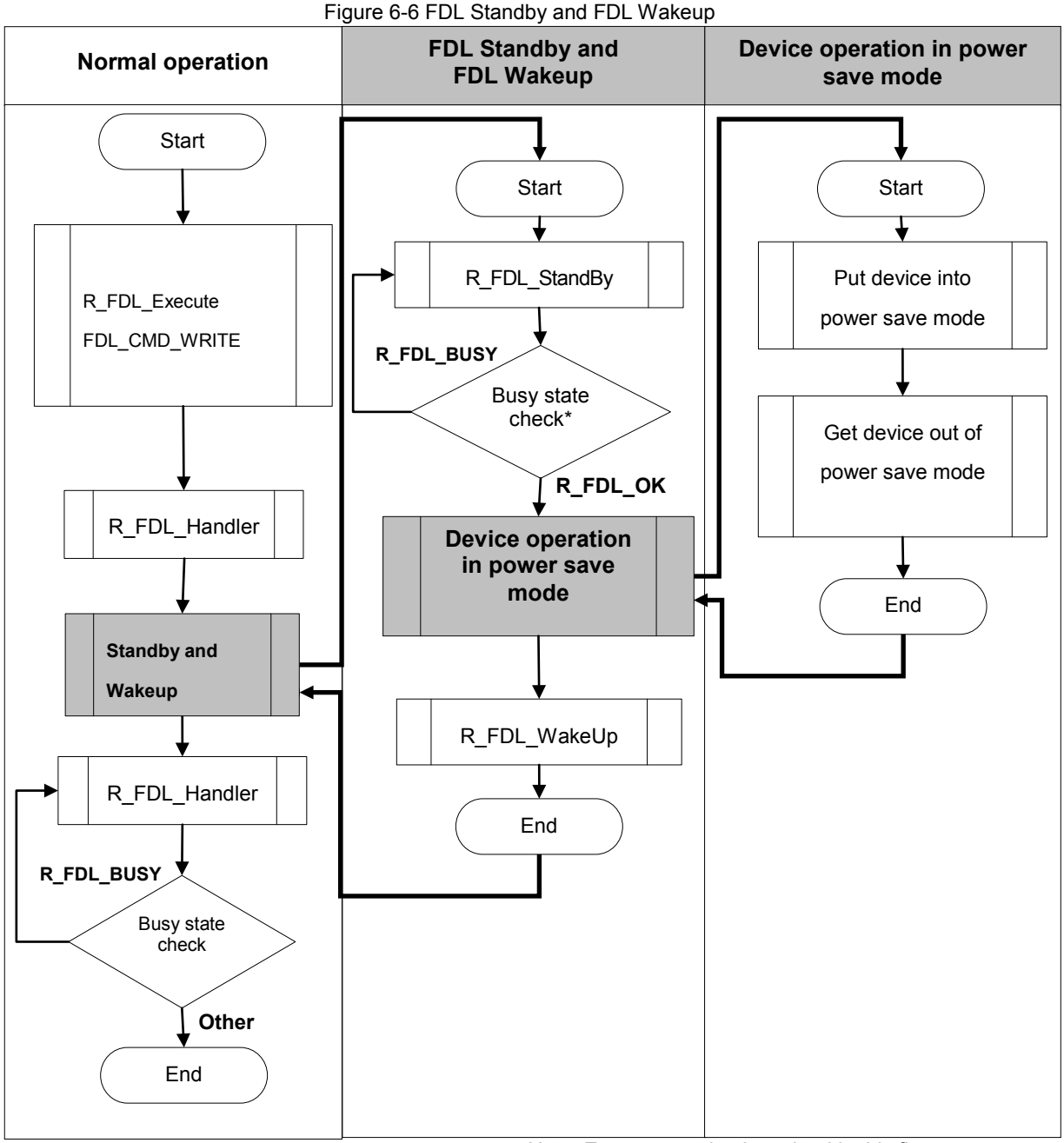
[Flowchart]

The figure below shows the basic flow of the functions, using the power save mode[Note1] of a device as an example. Busy state is checked using status_enu of the request structure .

Note 1: The HALT mode only. See table 4-1, Notes on FDL Architecture.

Note 2: Busy state is checked after execution of the R_FDL_StandBy function (see * in figure 6-6 below) using the return value of the function.

Figure 6-6 FDL Standby and FDL Wakeup



Note :Error processing is omitted in this figure

R_FDL_GetVersionString

Description

This function obtains the pointer to FDL version information.

C language format

```
(const uint8_t*) R_FDL_GetVersionString( void )
```

Pre-conditions

None

Post-conditions

None

Parameters

None

Return values

This function returns the following value (type definition const uint8_t*):

Value	Macro name
Address value	None

[Format for FDL version information]

"DH850T01xxxxxYZabcD"

- xxxxx = Compiler information
- Y = System reserved area
- Z = "E" Engineer version / "V" Release version
- abc = Library version
- D = Library version (only in the case of an engineer version)

Commands

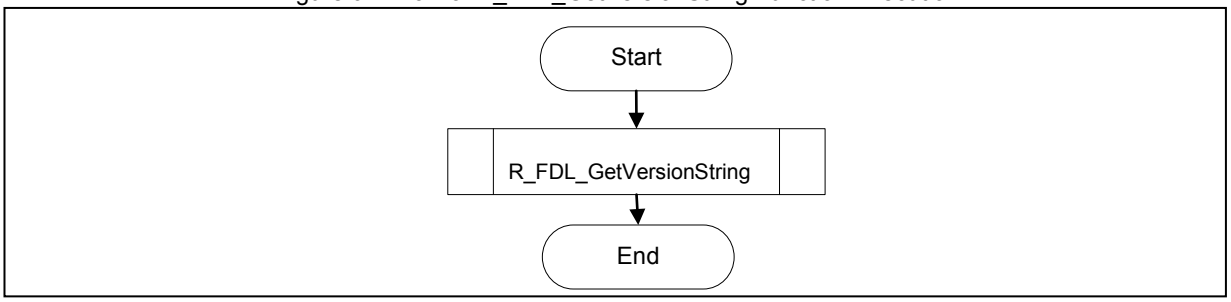
Reference

Settings

None

Flowchart

Figure 6-7 Flow of R_FDL_GetVersionString Function Execution



Chapter 7 FDL Processing Time

There are two types of FDL operation processing time: (1) FDL function response time and (2) hardware execution time [Note]. Figure 7-1 below shows the relationship between FDL function response time and hardware execution time. For details of hardware execution time, see the user's manual for the target device.

Note: Hardware execution time is not necessary for FDL functions (excluding R_FDL_StandBy) that do not need execution of the R_FDL_Handler function. The R_FDL_StandBy function is executed instead of the R_FDL_Handler function.

Figure 7-1 FDL Execution Sequence (FDL's Processing Time)

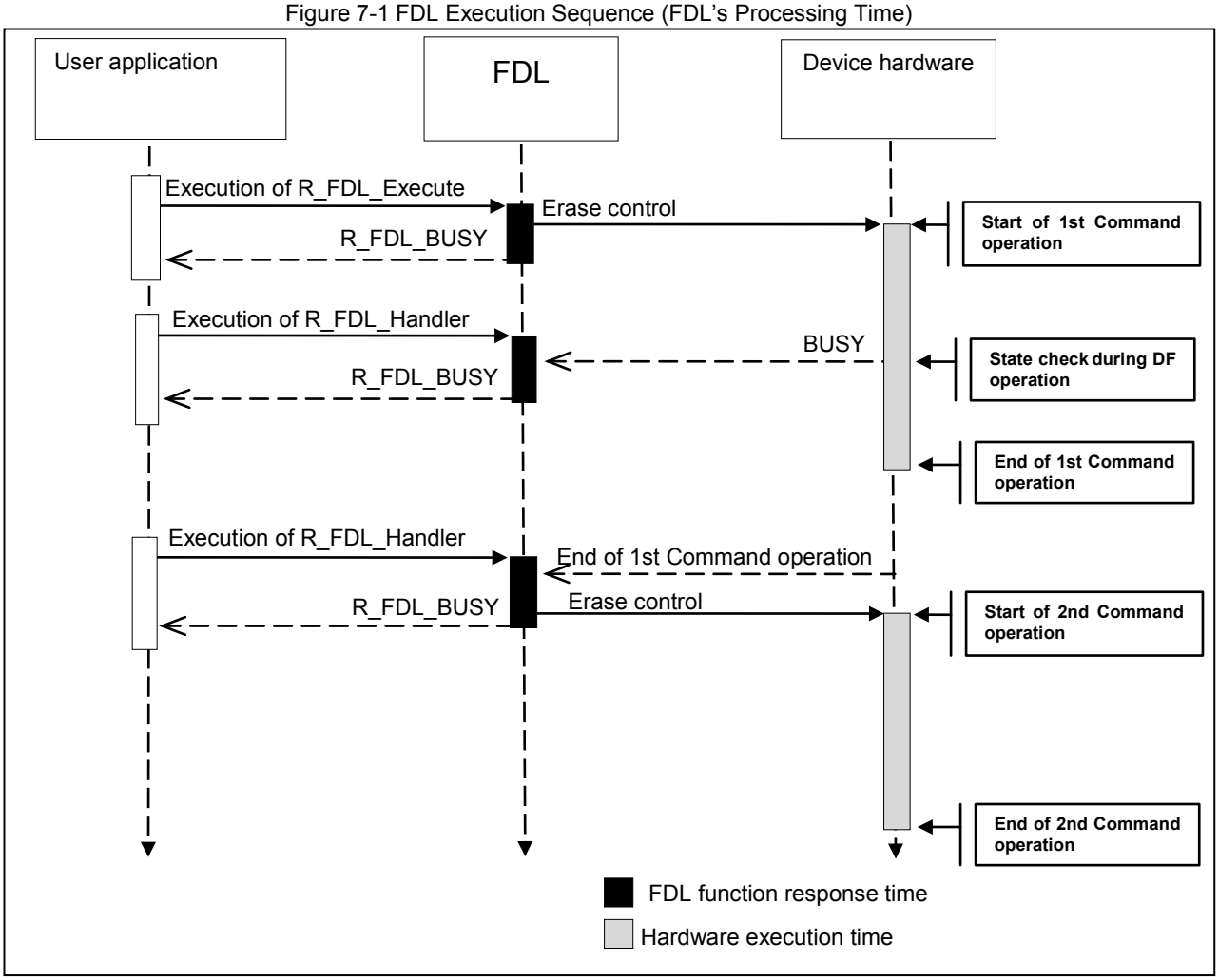


Table 7-1 shows typical response time of each FDL function.

<R> Table 7-1 FDL Function Response Time (Target: GHS-Version FDL V1.00)

FDL Function			Response Time [μs] Note
R_FDL_Init			51680/fxx
R_FDL_Execute	When R_FDL_CMD_ERASE is executed		480/fxx
	When R_FDL_CMD_WRITE is executed		640/fxx
	When R_FDL_CMD_BLANKCHECK is executed		480/fxx
	When R_FDL_CMD_READ is executed		320/fxx × word
R_FDL_Handler	R_FDL_Execute	When R_FDL_CMD_ERASE is executed	640/fxx
		When R_FDL_CMD_WRITE is executed	800/fxx
		When R_FDL_CMD_BLANKCHECK is executed	720/fxx

Note: fxx: CPU operating frequency (MHz), word: Set number of pieces of data (in units of 4 bytes)

Example) When the R_FDL_CMD_ERASE command of R_FDL_Execute (fxx = 80 MHz) is executed

480/80 = 6 -> the response time = 6 μs

For FDL execution time that is not described here, please contact our distributors or sales representatives.

Appendix A Revision History

Rev.	Date	Revisions	
		Page	Description
0.01	2013.03.11	—	First edition issued
1.00	2013.06.27	Throughout the document	Unified terms (e.g. request structure, command operation). Modified the style (e.g. correction of figure/table numbers)
			Deleted contents of figures and tables.
		7	Changed section 1.3, Terms, to a table.
		9	Changed the order in table 2-2 and added explanation and notes in section 2.3, Notes on Hardware Environment.
		10	Added sizes in section 3.2, FDL Resources.
		11	Added words in section 4.1, System Structure. Added figure 4-1, System Structure.
		12	Added explanation in table 4-1, Notes on FDL Architecture. Deleted unnecessary words in No.1.
		13	Divided command descriptions in section 5.1, List of FDL Functions.
		13	Created a table in section 5.2, List of Commands.
		15	Added explanation in section 5.4, Command Operation Flowchart.
		15	Added a new section “5.4.1 During Command Execution/During Busy.
		16	Added explanation in section 5.5, FDL Incorporation into the User Program.
		18	Changed words in figure 5-4, Example of Command Operation Sequence and simplified this figure.
		19	Added write processing to suspend target and changed the figure in section 5.7, Suspend/Resume.
		22	Added figure 5-6, Example of FDL Standby Operation Sequence.
		23	Added explanation in table 5-3, Notes on FDL Functions. Integrated notes on multiple execution to one item. Added explanation.
		24	Changed file names and macro names in section 6.1, Runtime Settings.
		25	Added explanation in table 6-1, Notes on Runtime Settings. Changed file names and macro names.
		26	Exchanged positions of figure 6-1 and figure 6-2 in section 6.2, Request Structure Settings. Added explanation and changed definition.
		28	Added explanation in section 6.2.2, User Read Access.
		28	Added 3 and 4 in table 6-2, Notes on Request Structure Settings.
		31, 32	Deleted unnecessary words in the R_FDL_Init function. Added pre-conditions. Added category and description in “Return value”. Modified figure 6-4. Changed function names and file names. Added supplementary information.
		34～36	Added status_enu used by the R_FDL_Handler function in “Reference”. Added category and description. Added status_enu used when R_FDL_SuspendRequest or R_FDL_ResumeRequest is executed.
		40	Added notes on post-conditions in R_FDL_Handler in table 6-2.

Rev.	Date	Revisions	
		Page	Description
1.00	2013.06.27	41, 42	Added “write” to the target of R_FDL_SuspendRequest. Added category and description in “Return value” and “Reference”. Shifted R_FDL_SUSPENDED to the page of R_FDL_Execute. Added other information.
		43	Added category and description in “Return value” and “Reference” of R_FDL_ResumeRequest. Shifted R_FDL_BUSY to the page of R_FDL_Execute.
		45, 46	Added explanation of R_FDL_StandBy. Added category and description in “Return value” and “Reference”.
		47	Added category and description in “Return value” and “Reference” of R_FDL_WakeUp.
		50	Added chapter 7, FDL Processing Time.
1.01	2013.07.31	10, 52	Added target products and versions in the table.
		18	Added “n = 1, 2, 3, ---” in figure 5-4, Example of Command Operation Sequence.”
		19	Deleted unnecessary words.
		23,24	Divided table 5-3, Notes on FDL Function. Modified words and added supplementary information.

RH850 Family User's Manual
Type 01 Data Flash Library

Publication Date: Sep 30, 2013 Rev.1.01

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2013 Renesas Electronics Corporation and Renesas Solutions Corp.

Colophon 1.3

RH850 Family

Renesas Electronics Corporation

R01US0063EJ0101